CRYPTOGRAPHIC MODULAR EXPONENTIATION METHOD PROTECTED AGAINST DPA ATTACKS

In the field of the protection of cryptographic algorithms against DPA attacks, the invention concerns a method during which a modular exponentiation of the type x^d is carried out, with d an integer exponent of m+1 bits, scanning the bits of d from left to right in a loop indexed by i varying from m to 0 and calculating and storing in an accumulator (R0), at each turn of rank i, an updated partial result equal to x^b(i). b(i) corresponds to the m-i+1 most significant bits of the exponent d: b(i) = $d_{m->i}$. The number consisting of the bits of weights j to k of d is defined by: $d_{k->j} = (d_k,...,d_j)_2 = \sum_{i=j}^{k} d_i.2\hat{}(i-j)$.

Modular exponentiation is one of the elementary operations used in many cryptosystems, such as RSA (Rivest, Shamir and Adleman) cryptosystems or DH (Diffie and Hellman) cryptosystems. For such

applications, x is for example a message to be enciphered or deciphered, to be signed or authenticated, and d is for example a public key, a secret key or part of such a key.

Since the invention of public key cryptography by Diffie and Hellman, many public key cryptosystems have been proposed. Amongst those which resist cryptographic analysis, the RSA cryptosystem is without any doubt the most widely used. Its intrinsic security lies in the difficulty in factorising large integer numbers. Despite intensive researches, the problem of factorisation is still considered to be a significant problem, making the RSA cryptosystem secure for sensitive applications such as for example the enciphering of data or digital signature.

Thus, rather than attempting to break the RSA algorithm at a mathematical level, cryptographs have become interested in the concrete implementations of the RSA cryptosystems. This has led to a huge increase in fault attacks and covert channel attacks, aimed at discovering a particular confidential information (such as for example keys or parts of keys) manipulated during one or other the steps implemented by the calculation device executing a cryptographic operation.

The most widely known covert channel attacks are said to be simple or differential. Simple (SPA or differential (DPA) covert channel attack means an attack based on the measurement of a physical quantity from the outside of the device, whose direct analysis (simple attack SPA) or analysis according to a

statistical analysis (differential attack DPA) makes it possible to discover information manipulated in the device. These attacks were in particular disclosed by Paul Kocher (Advances in Cryptology - CRYPTO'99, vol. 1666 of Lecture Notes in Computer Science, pp.388-397, Springer-Verlag, 1999).

Amongst the physical quantities that can be exploited for these purposes, it is possible to cite the execution time, the current consumption, the electromagnetic field radiated by the part of the component used for executing the calculation, etc. These attacks are based on the fact that, during the execution of an algorithm, the manipulation of a bit, that is to say its use by a particular instruction, leaves a particular impression on the physical quantity considered, according to the value of this bit and/or according to the instruction.

There exist two families of implementations of exponentiation algorithms making it possible to evaluate the value of $y = x^d \bmod N$: the so-called right to left implementations and the so-called left to right implementations.

In the left to right implementations, the bits of the exponent are scanned from the most significant bit to the least significant bit. In this second family of exponentiation algorithm there is in particular known the SAM algorithm (standing for Square and Multiply) and its variants such as sliding window algorithms. Compared with the so-called right to left algorithms, left to right algorithms require less memory and allow

the use of precalculated powers x^i in order to accelerate the calculation of y. All the left to right algorithms have in common the use of an accumulator (or register) that is updated throughout the calculation in order to store the value of $x^{d_{m->i}}$ mod N for decreasing values of i until the accumulator contains the final value $y = x^{d_{m->0}} = x^d$ mod N. $d_{k->j}$ is the word consisting of the bits of weight j to k of d.

The general principle of the SAM algorithm is as follows. The bit of weight i of d is denoted $d = (d_m,...,d_0)_2 = \sum_{i=0}^{m} d_i.2^i$, the binary representation of the exponent d, with $d_i \in \{0, 1\}$. For each bit of d, the SAM algorithm stores in an accumulator (register R0) an updated result calculated from the recurrence equation $x^{d_{m->i}} = (x^{d_{m->i+1}})^2 * x^{d_i}$, with $x^{d_{m->m}} = x^{d_m}$, which is summarised by the following algorithm:

```
Input:   x, d = (d_m,...,d_0)_2
Output:  y = x^d mod N
    R0 <- 1; R2 <- x, i <- m
    as long as i ≥ 0, do:
        R0 <- R0xR0 mod N
        if d_i = 1 then R0 <- R0xR2 mod N
        i <- i-1
    end as long as
    return R0
```

R0 <- x means that the value of x is stored in the register R0. R0xR0 means that the content of the register R0 is squared. R0xR2 means that the content of the register R0 is multiplied by the content of the

register R2. Finally, $d_{i->j}$ refers to the bits of rank j to i of d.

In order to guard against implementation attacks, it is known that it is necessary to make the algorithms random. In the case of the RSA cryptosystem, two types of countermeasure are currently known for making the calculation of y = x^d mod N random.

The first type of countermeasure consists of making the input data of the algorithm random.

A first example of this first countermeasure consists of making the data item x random before carrying out the modular exponentiation, by adding to x a random term and making the calculations modulo 2^k N, before a final modulo N:

$\bar{x} <- x + r1.N$, with r1 a random number of k bits and make the calculations modulo (2^k).N, before a final reduction modulo N. This first countermeasure, described by P Kocher, has the advantage of being independent of the exponentiation algorithm.

A second example of this first countermeasure consists of the making the exponent d random before carrying out the modular exponentiation, by adding to it a random term:

$\bar{d} <- d + r2.\phi(N)$, r2 a random number of k bits.

Usually these two solutions are combined in order to perform the operation $y = \bar{y}$ mod N with $\bar{y} = \bar{x}^{\wedge}\bar{d}$ mod $(2^k.N)$.

In a third example of this first countermeasure used alone, for example when x is the result of a

probabilistic formatting (for example using the PSS or Probabilistic Signature Scheme function), since in this case x is already masked and a direct calculation is made of $y = x^{\overline{d}} \bmod N$ with $\overline{d} = d + r2.\phi(N)$ with r2 random.

Unfortunately, such a randomisation of the exponent d is limited to particular implementations, called CRT implementations, of the RSA cryptosystem since the value of Euler's constant $\phi(N)$ is not generally known from the private exponentiation algorithm in its standard version (that is to say not CRT).

The second countermeasure consists of making the exponentiation algorithm itself random. The best putting into practice of the second countermeasure is the MIST algorithm of Walter. The MIST algorithm randomly generates a new addition chain for the exponent d in order to effect $x^d \bmod N$. In order to minimise the number of registers, the addition chain is carried out on the fly by means of an adaptation of an exponentiation algorithm based on division chains. Another example is an improved version of a sliding window algorithm (see Kouichi Itoh, Jun Yajima, Masahiko Takenaka and Naoya Torii, DPA countermeasures by improving the window method CHES 2002, volume 2523 of Lecture Notes in Computer Science, pages 303-317, Springer Verlag 2002). Compared with the first countermeasure, this makes it possible to make the exponentiation random without needing to know $\phi(N)$ but

requires a secure division algorithm for calculating the division chains and causes not insignificant concerns about management of the calculations.

The invention proposes a novel method for making random the execution of a modular exponentiation, for the purpose of guarding against differential attacks (DPA), having the advantages of the two known countermeasures: as in the first countermeasure, the method according to the invention does not impose any particular exponentiation algorithm and applies to any exponentiation algorithm, and as in the second countermeasure, in the invention, the algorithm itself is made random, not only the data that it manipulates. Thus the algorithm does not need to known $\phi(N)$ and/or the public key e in an RSA exponentiation (the key e is often unavailable to the signature or deciphering algorithm).

The method according to the invention introduces the concept of auto-random exponentiation, meaning that the exponent d is used itself as an additional source of randomness in the exponentiation process.

Thus the invention concerns a cryptographic method during which a modular exponentiation of the type x^d is carried out, with d an integer exponent of m+1 bits, by scanning the bits of d from left to right in a loop indexed by i decremented from m to 0 in steps of 1 and calculating and storing in an accumulator, at each turn of rank i, an updated partial result equal to x^b(i), b(i) being the m-i+1 most significant bits of the exponent d.

The method according to the invention is characterised in that:

- at the end of a turn of rank i(j) (i = i(0)) chosen randomly, a randomisation step E1 is performed during which:

E1: a random number z (z = b(i(j)), z = b(i(j)).2^{τ}, z = u) is subtracted from a part of the bits of d not yet used ($d_{i-1->0}$) in the method

- then, after having used the bits of d modified by the randomisation step E1, a consolidation step E2 is performed during which:

E2: the result of the multiplication of the content of the accumulator (x^b(i)) by a number that is a function of x^z stored in a register (R1) is stored (R0 <- R1xR0) in the accumulator R0.

From a practical point of view, during step E1, the number z is subtracted from the content of a register in which the exponent d is initially stored, and the result of the subtraction is stored in the same register, and then the bits of b are continued to be scanned.

So that the result of the exponentiation x^d mod N is correct at the end of the method, the randomisation step E1 must not modify the bits of d already used in the calculation (it will be recalled that the method uses a left to right algorithm). The index i(j) at which the randomisation E1 is carried out, chosen randomly, must be chosen so that the m-i(j)+1 most significant bits of the register containing initially

the exponent d remain unchanged during step E1. This condition will be referred to hereinafter as a "consistency" condition.

The essential idea of the invention is thus to use a chopping of the calculation of x^d mod N of the form: $x^d = x^{(d-z)} * x^z$ (described in French patent application number 02 04117 (number to be confirmed)) with z a random number used as a means of masking the exponent d. Preferably appropriate values of z are chosen such that x^z can be obtained easily from x^b already calculated moreover during the method. It should be noted that a totally random choice of z gives rise to an almost doubling of the calculation time.

The method according to the invention applies independently of the left to right exponentiation algorithm. Moreover, the rank i(j) at which step E1 is performed is chosen so as to be random, and therefore the method itself is random, and not only the data that it manipulates.

The method according to the invention is also effective in terms of space (it requires only one additional calculation register) and in terms of calculation time, as will be seen better subsequently in the example of the SAM algorithm.

The method according to the invention is also easy to implement whatever the algorithm to which it is applied. It is not based on any group property and its implementation does not require previously knowing the order of the group in which the exponentiation is performed.

Finally, the method according to the invention can be used conjointly with other algorithm protection measures, such as the countermeasures disclosed by P Kocher and stated above.

The randomisation step E1 can be performed on a single occasion during the method. A step E1 can also be performed several times, at the end of various turns of rank i(j) (that is to say at rank i = i(0), and then at rank i = i(1), ..., and then finally at rank i = i(f)) chosen randomly between 0 and m. The idea is here to improve further the security of the method by using the equation:

$$x^{\wedge}d = x^{\wedge}(d-z1-z2-\ldots-zf) \times x^{\wedge}z1 \times x^{\wedge}z2 \times \ldots \times x^{\wedge}zf$$

$$= x^{\wedge}(((d-z1)-z2)-\ldots-zf) \times ((x^{\wedge}z1) \times x^{\wedge}z2) \times \ldots \times x^{\wedge}zf$$

It is possible to choose, at the start of the method, the random rank or ranks i(j) at which a randomisation E1 is performed. For example, at the start of the method, a predefined set {i(0), i(1), ..., i(f)} of f+1 (f being random or not) values of the index i for which it is wished to perform a randomisation E1 is determined. In this case, at each turn, it is decided or not to perform a randomisation E1 according to whether or not the current index i forms part of the predefined set.

It is also possible to choose randomly at the start of each turn i to perform or not the randomisation step E1. In this case, a Boolean variable ρ is for example used, drawn randomly at the end of each turn of index i.

Different embodiments of the invention will now be described, which differ from one another essentially by the embodiment of step E1, and in particular by the choice of z and the choice of the part of d from which is subtracted.

According to a first embodiment, it is chosen to perform the consolidation step only once at the end of the method. This makes it necessary to subtract z routinely from the least significant bits of the exponent d, so as to obtain a correct result at the end of the method.

According to a first variant of this embodiment, $z = b(i(j)) = d_{m->i(j)}$ is chosen for a chosen random number i(j) and, during the randomisation step E1, b(i(j)) is subtracted from d, that is to say from the least significant bits of d.

The choice z = b(i(j)) is particularly advantageous since $x^{b(i(j))} = x^{d_{m->i(j)}}$ is already available in the accumulator at the end of the turn i(j) and therefore does not need to be calculated. The variable i(j) is chosen so that the bits of weight i(j) to m of the number d-b(i(j)) are equal to the bits of weight i(j) of the number d, so that the first m-i(j)+1 turns of the calculation x^d are identical to the first m-i(j)+1 turns of the calculation of the $x^{(d-b(i(j)))}$ (consistency condition). At the end of the turn i(j), d-z = d-b(i(j)) is calculated and the content of the accumulator x^b is stored in the register (E1).

In a particular example, a Boolean variable ρ is used to determine, at the end of each turn of index i, whether or not a randomisation is performed. If ρ takes an active value, then step E1 is performed: the number d is replaced with the number d-b(i(j)) and x^b(i(j)) is stored.

As in the conventional left to right algorithm, the accumulator R0 is used to store the value of $x^{d_{m->i}}$, at each turn of index i. The register R1 is used for storing the product: $\Pi_j x^{d_{m->i(j)}}$.

The whole applies to the known SAM algorithm, and the following algorithm I is obtained:

Input: x, $d = (d_m, \ldots, d_0)_2$

Output: $y = x^d \bmod N$

```
R0 <- 1; R1 <-1; R2 <- x, i <- m
as long as i ≥ 0, do:
    R0 <- R0xR0 mod N
    if d_i = 1 then R0 <- R0xR2 mod N
    ρ <- R{0, 1}
    if ((ρ = 1) and d_{i-1→0} ≥ d_{m→i} then
        d ← d - d_{m->i}
        R1 <- R1xR0 mod N
    end if
    i <- i-1
end as long as
R0 <- R0xR1 mod N
return R0
```

ρ <- R{0, 1} means that the value of ρ is chosen randomly in the set {0, 1}. ρ is thus a random Boolean variable.

The randomisation step E1 (d <- d - $d_{m->i(j)}$; R1 <- R1xR0 mod N) is performed only if ρ = 1 (that is to say if a randomisation must be performed) and if $d_{i(j)-1->0} \geq d_{m->i(j)}$.

The condition $d_{i(j)-1->0} \geq d_{m->i(j)}$ means that the bits of weight 0 to i-1 of d are greater than b(i(j)), b(i(j)) being equal to the bits of weight i(j) to m of d. This guarantees that the m-i+1 most significant bits of d-b(i(j)) are identical to the m-i+1 most significant bits of d, and therefore that the first m-i+1 calculation turns of x^d are identical to the first m-i+1 calculation turns of x^(d-b(i(j))).

The "consistency" condition ($d_{i(j)-1->0} \geq d_{m->i(j)}$) means that only the least significant bits of the exponent d are made random. In addition, it will be noted that the randomisation step d <-d - $d_{m->i(j)}$ modifies only the (m-i(j)+1) least significant bits of d.

It should be noted that, in the algorithm I, as at the iteration i = i(j) the updating step d <- d - $d_{m->i}$ does not modify the (m-i+1) most significant bits of d, this step can be replaced by the equivalent step:

$d_{i-1->0}$ <- $d_{i-1->0}$ - $d_{m->i}$.

According to a second variant of the first embodiment, z is chosen equal to g.b(i), with g a random number such that $d_{i(j)-1->0} \geq g.d_{m->i(j)}$. In this

case the equation $x^d = x^{(d-z)}.x^z = x^{(d-g.b(i))}.(x^{b(i)})^g$. is used, and, from a practical point of view, in order to perform a randomisation E1 at the end of the turn of index i(j):

- z = g.b(i) is calculated and the result is subtracted from the exponent d,

- the register R1 is updated by multiplying its content by the content of the accumulator ($x^{b(i)}$) exposed to the power g. Which can in concrete terms be implemented by the instruction R1 <- R1xR0^g mod N.

Preferably $g = 2^\tau$ is chosen, τ being a random integer. This considerably simplifies the calculation since the calculation of $g.b(i) = g.d_{m->i(j)}$ amounts to a simple shifting of bits and the evaluation of $(x^{b(i)})^g$ mod N amounts to performing the calculation of τ squared.

Since multiplying by 2^τ amounts to a shifting of bits, the instruction $d <- d - 2^\tau.d_{m->i}$ which calculates $d-g.b(i)$, can be replaced by $d_{m->\tau} <- d_{m->\tau} - d_{m->i}$, or better by the equivalent instruction $d_{i-1->\tau} <- d_{i-1->\tau} - d_{m->i}$

In addition, as for the other embodiments, it must be verified that, at the iteration i=i(j), $d_{i-1->0} \geq 2^\tau.d_{m->i}$. This condition of consistency can be replaced by an equivalent but more effective test: $d_{i-1->\tau} \geq d_{m-i}$.

Preferably, τ is chosen randomly in the set {0, ..., T}. The delimiter T is chosen as the best compromise between the randomisation of the most

significant bits of d and the efficacy (in terms of calculation time in particular) of the calculation of the τ squared.

In the particular example of the SAM algorithm, the following algorithm I' is finally obtained:

```
Input:   x, d = (d_m,...,d_0)_2
Output:  y = x^d mod N
    R0 <- 1; R1 <- 1; R2 <- x, i <- m
    as long as i ≥ 0, do:
        R0 <- R0xR0 mod N
        if d_i = 1 then R0 <- R0xR2 mod N
        ρ <- R{0, 1}; τ <- R{0, ..., T}
        if ((ρ = 1) and (d_{i-1→τ} ≥ d_{m-i})) then
            d_{i-1→τ} ← d_{i-1→τ} - d_{m->i}
            R3 <- R0
             as long as (τ > 0) do:
                R3 <- R3^2 mod N; τ <- τ-1
            end as long as
            R1 <- R1xR3 mod N
        end if
        i <- i-1
    end as long as
    R0 <- R0xR1 mod N
return R0
```

One advantage of the algorithm I' is that the top half of d is partly randomised and consequently entropy (that is to say randomness) is added. On the other hand, an additional register R3 is necessary for calculating $R0^{2^\tau}$.

Algorithms I and I' may be sufficient to protect the exponents in certain cases. For example, because of its construction, the RSA cryptosystem always exposes the most significant half of the private exponent d if the corresponding public exponent is small. Making random the most significant bits of d would therefore afford no protection for such an algorithm.

However, for other algorithms and in other situations, making all the bits of d random would afford additional security.

For this purpose, it is proposed, in a second embodiment, to choose $z = b(i(j)) = d_{m->i(j)}$ for a random number i(j) and, during step E1, b(i) is subtracted not from d but from some of the bits of d corresponding to the bits of d of weight i(j)-c(j) to i(j)-1, c(j) being a integer number such that $i(j) \geq c(j) \geq 0$. This can be expressed by the following instruction:

$$d_{m->i(j)-c(j)} <- d_{m->i(j)-c(j)} - d_{m->i(j)}$$

Preferentially, as during step E1 of a randomisation to rank i(j), the bits of weight i(j) - c(j) to i(j) - 1 of d are modified and it is chosen only to perform one randomisation at a time, and it is chosen to perform a consolidation step at the end of the rank using the last bit of d modified during the previous randomisation step E1 (rather than at the end of the method), that is to say after the evaluation of the partial result $x^{(d_{m->i(j)-c(j)})}$ mod N.

This amounts to imposing the condition $i(j+1) \leq i(j) - c(j)$, i(j+1) being the index of the following

randomisation. This makes it possible not to use additional registers to store the bits of the exponent that were modified during a previous randomisation. Equally, $i(j) - c(j) < 0$ is chosen, so that $i(j) - c(j)$ can be used to define the rank of a bit of d for calculating $x^{(d_{m->i(j)-c(j)})} \bmod N$.

These two conditions can be made concrete by the use of a Boolean semaphore σ which indicates whether an update is authorised or not: σ has an inactive value as long as $i \geq i(j) - c(j)$ and is activated when $i < i(j) - c(j)$. It also becomes unusable as soon as $i(j) - c(j) < 0$.

The $(m-i(j)+1)$ most significant bits of d remain unchanged during the randomisation step if (consistency condition):

$$d_{i(j)-1 \to i(j)-c(j)} \geq d_{m \to (j)}$$

$$(i(j)-1)-(i(j)-c(j)) \geq m-i(j) \Leftrightarrow c(j) \geq m-i(j)+1$$

According to a first variant of the second embodiment, $c(j)$ is chosen equal to $m-i(j)+1$. With the condition $i(j) \geq c(j) \geq 0$, the condition $c(j) \geq m - i(j) + 1$ is satisfied if $2.i(j) \geq m+1$.

The modified SAM algorithm according to this variant can therefore be written (algorithm II):

Input: x, $d = (d_m, \ldots, d_0)_2$

Output: $y = x^d \bmod N$

R0 <- 1; R1 <-1; R2 <- x,

i <- m; c <- -1; σ <- 1

as long as $i \geq 0$, do:

R0 <- R0xR0 mod N

```
            if d_i = 1 then R0 <- R0xR2 mod N end if
            if (2i ≥ m+1) and (σ=1) then c <- m-i+1
                                    if not σ = 0
            end if
            ρ <- R{0, 1}
            ε <- ρ and (d_{i-1 ->i-c} ≥ d_{m->i}) and σ
            if ε = 1 then
                R1 <- R0; σ <- 0
                d_{i-1 -> i-c} <- d_{i-1 -> i-c} - d_{m->i}
            end if
            if c = 0 then R0 <- R0xR1 mod N; σ <- 1
            end if
            c <- c-1; i <- i-1
        end as long as
    return R0
```

It should be noted that algorithm I corresponds to algorithm II in the case where c(j) = i(j) for any j. It should also be noted that, in algorithm II, the consistency condition ($d_{i(j)-1 \to i(j)-c(j)} \geq d_{m \to i(j)}$) is satisfied during the first part of the algorithm, considering approximately that $d_{i(j)-1 \to i(j)-c(j)}$ and $d_{m \to i(j)}$ are random numbers of (m-i(j)+1) bits. It should be noted, in this algorithm, that all the bits of the exponent are randomised.

According to a second variant of the second embodiment, c(j) is chosen randomly and between i(j) and m-i(j)+1.

It was seen previously that the condition c(j) ≥ m - i(j) + 1 must be satisfied. By posing c(j) = m -

$i(j) + 1 + \nu(j)$, it is therefore necessary to satisfy $\nu(j) \geq 0$. Moreover, with the condition $i(j) \geq c(j) \geq 0$, there comes $2.i(j) \geq m+1+\nu(j)$. Therefore the greatest possible value for $\nu(j)$ is $2.i(j)-m-1$ and therefore, as $\nu(j) \geq 0$, the parameter $c(j) = m-i(j)+1+\nu(j)$ can take any value in the set $\{m-i(j)+1, \ldots, i(j)\}$. The algorithm II can then be generalised by choosing $c(j)$ random in the set $\{m-i(j)+1, \ldots, i(j)\}$.

In the particular example of algorithm II this amounts to replacing the instruction:

if $(2i \geq m+1)$ and $(\sigma = 1)$ then $c \leftarrow m-i+1$

by the instruction:

if $(2i \geq m+1)$ and $(\sigma=1)$ then $c \leftarrow R\{m-i+1, i\}$

which gives the following algorithm III:

Input: x, $d = (d_m, \ldots, d_0)_2$

Output: $y = x^{\wedge}d \bmod N$

$R0 \leftarrow 1$; $R1 \leftarrow 1$; $R2 \leftarrow x$,

$i \leftarrow m$; $c \leftarrow -1$; $\sigma \leftarrow 1$

as long as $i \geq 0$, do:

$R0 \leftarrow R0 \times R0 \bmod N$

if $d_i = 1$ then $R0 \leftarrow R0 \times R2 \bmod N$ end if

if $(2i \geq m+1)$ and $(\sigma = 1)$

then $c \leftarrow R\{m-i+1, i\}$

if not $\sigma = 0$

end if

$\rho \leftarrow R\{0, 1\}$

$\varepsilon \leftarrow \rho$ and $(d_{i-1 \rightarrow i-c} \geq d_{m \rightarrow i})$ and σ

if $\varepsilon = 1$ then

```
                R1 <- R0; σ <- 0
                d_{i-1 -> i-c} <- d_{i-1 -> i-c} - d_{m->i}
            end if
            if c = 0 then R0 <- R0xR1 mod N; σ <- 1
            end if
            c <- c-1; i <- i-1
        end as long as
    return R0
```

A large value for $\nu(j)$ increases the probability of success for the consistency condition (and therefore the choice of a randomisation). On the other hand, this also reduces the possible values of the index i satisfying the condition $2.i(j) \geq m+1+\nu(j)$.

The frequency of occurrence of the value $\rho = 0$ of the Boolean variable ρ is a parameter of the method making it possible to choose the best compromise between performance and security, according to the application envisaged: the more randomisation steps are performed, the greater the detriment to the total calculation time; conversely, the fewer randomisation steps are performed, the more attacks by exhaustive search are facilitated.

A good means for minimising the cost of the additional operations consists of slightly modifying the random number generator producing the number ρ so that, when the Hamming weight of d-z (z can have different values as a function of b(i), according to the embodiment envisaged) is lower than the Hamming weight of d, ρ has a greater probability of equalling

1, and vice versa. With this trick, the algorithm will tend to select the case having the lowest Hamming weight, that is to say the most rapid branch.

It should be noted only that this algorithm cannot always select the most rapid branch, otherwise it would become deterministic and therefore easily attackable.

According to a third embodiment of the invention, a random number u of v bits is chosen at the start of the method and x^u is stored in the register R1. The number u is preferably modified several times during the method, in order to increase the random factor in the method.

Then, during the calculation, for a given rank i(j), it is wondered, for a packet w of v bits of d such that w>u, whether the calculation x^w is more expensive (in terms of calculation time) then that of x^(w-u)*x^u.

To reply to this question, it suffices to determine whether H(w) > H(w-u) + 1. H(w) is the Hamming weight of w and represents the cost of the operation x^w. H(w-u) is the Hamming weight of x^(w-u), representing x^(w-u). The term "+ 1" represents the cost of the multiplication of x^(w-u) by x^u (x^u also being stored).

If the calculation of x^w is less expensive than the calculation of x^(w-u)*x^u, then the method is continued. Otherwise, if the calculation of x^w is more expensive than the calculation of x^(w-u)*x^u, then the

packet w of bits of d is replaced by the number w-u. The consolidation step (here a multiplication by x^u, which is represented by the operation R0 <- R0xR1 mod N) will be performed when all the modified bits of d have been used.

Compared with the two previous embodiments described, this third embodiment has the advantage of being faster since, in order to effect a randomisation, the most rapid path (the least expensive) is chosen each time. Thus it is shown experimentally that the complexity of this method is approximately 1.4. The complexity is the average number of multiplications of contents of registers performed for each bit of the exponent d. The complexity of an unprotected SAM algorithm is 1.5; the complexity of the methods according to the first or second embodiments of the invention is for its part slightly greater than 1.5.

Moreover, in this third embodiment, the source of randomness (the number u) is external to the method. Finally, the resources (in particular the number of registers) used are the same.

This third embodiment can be represented concretely by the following algorithm IV:

Input: x, d = $(d_m,\ldots,d_0)_2$

Parameters: v, k

Output: y = x^d mod N

R0 <- 1; R2 <- x; i <-m; $L = \{\}$

as long as i ≥ 0, do:

R0 <- R0xR0 mod N

if d_i = 1 then R0 <- R0xR2 mod N end if

```
                if i = m mod ((m+1)/k)) then σ<-1 end if
                if σ = 1 and L = {} then
                   (modification of the number u during the method)
                   σ <- 0: u <- R {0, ..., 2^v-1};
                   R1 = x^u mod N
                end if
                w <- d_{i->i-v+1}
                h <- H(w)
                if w ≥ u then Δ <- w-u; h_Δ <- 1 + H(Δ)
                         if not h_Δ <- v+2
                end if
                ρ <- R{0, 1}
                if [(σ=0)∧(i-v+1≥0)] ∧
                    [(h>h_Δ) or ((ρ=1) and (h=h_Δ))] then
                 (it is chosen to effect x^(w-u))
                   d_{i->i-v+1} <- Δ; L <- L ∪ {i-v+1}
                end if
                if (i ε L) then
                R0 <- R0xR1 mod N
                L <- L\{i}
                end if
                i <- i-1
            end as long as
        return R0
```

In this example, the set L contains the list of indices for which a consolidation step must be performed. The instruction "if d_i = 1 then R0 <- R0xR2 mod N end if" is the conventional instruction of an SAM algorithm, performed for each value of i.

The exponent d is here divided into k blocks, of identical sizes if m+1 is divisible by k or of identical sizes to within one unit otherwise.

At each start of a block (that is to say for i = m mod((l+1)/k)), the variable σ is set to 1. Next, when σ is equal to 1, it is necessary to wait for the set *L* to be empty before performing a new randomisation set. At each consolidation step, a corresponding index i(j) is removed from the set *L* (instruction *L* <- *L*\{i}). When the set *L* is empty, a new value of u can be chosen and x^u is calculated by means of a conventional SAM algorithm using the registers R1 and R2.

At the middle of each block (σ = 0), one or more randomisation steps are performed, when $h>h_\Delta$ or (ρ=1 and $h = h_\Delta$), and on each occasion (instruction *L* <- *L* ∪ {i-v+1}) the index $i(j)^{-v+1}$ at which it will be necessary to perform a consolidation step is stored. It is therefore necessary for i-v+1 to be a valid consolidation index, that is to say i-v+1 ≥0 (consistency condition). At each randomisation step, if $h > h_\Delta$, it is chosen to perform the operation x^(w-u)*x^u, which is less expensive, and the bits of d are modified accordingly ($d_{i->i-v+1}$ <- Δ). If $h < h_\Delta$, it is chosen to effect x^w, which is less expensive, and d is not modified. If $h = h_\Delta$, it is chosen randomly (ρ = 0 or 1 random) to effect x^(w-u)*x^u or x^w.

By way of indication, the average number of modular multiplications necessary for performing an exponentiation of length 1024 with the SAM algorithm, protected or not, is given:

- SAM without protection: 1536 multiplications
- SAM protected by adding a multiple of $\Phi(n)$ ($r.\Phi(n)$ with r of 64 bits) added to the exponent d (prior art): 1536 + 96 multiplications
- SAM protected according to algorithm II or III: 1536 + 10 multiplications
- SAM protected according to algorithm I': 1536 + 512 multiplications. $\overline{\rho}, \overline{\rho}$ being the average value of ρ
- SAM protected according to algorithm IV: 1443 multiplications

It is seen through these examples that a protected algorithm according to the invention is very effective, in terms of multiplications performed (and therefore calculation time).